Physical travelling salesman problem solver

by Rok Šibanc
Slovenia, Europe

rok.sibanc@gmail.com

1 PTSP solver

PTSP solver = PATH_solver + VECTOR_evolver

I divided PTSP problem into two separated tasks. First one was to find the sequence of cities in which the travelling salesman should visit them and second one was to find the fastest path between these sequence of cities. Fastest path is the path with lowest number of force vectors of course. There were a lot of experiments with different fitness functions and with population size with both solvers.

Solvers were programmed in C++ and ran on Pentium 4, 3.06GHz with 512Mb RAM.

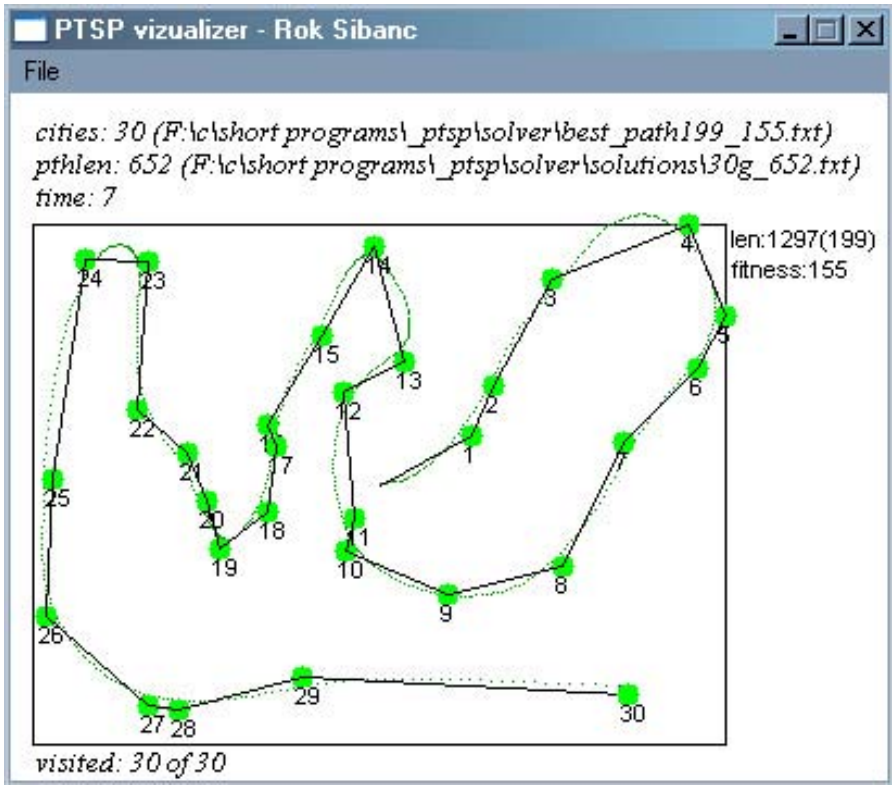

Figure 1 : best solution (652 vectors)

1.1 PATH_solver

TSP with a twist!

The path solver was somehow a replica of normal TSP solver but with a small twist. The fastest path is not neccessarily the shortest one, because of the rules of moving the traveling salesman. So how to include the curvature of the path to the fitness function of the TSP?
The answer is the angles between the cities can somehow help us to develop a good fitness function for our PATH_solver.

The fitness function:
sumcosangdist+=(sqrtdist1+sqrtdist2)(k+cosang);*

- *sqrtdist1* and *sqrtdist2* are square roots of the lenght between three cities
- *cosang* is the cosine of angle between these three cities.
- *k* is a constant which determines whether the path lenght or path curvature is more important. The best values of *k* were around 3.

Evolutionary algorithm:
Genome: numbers 1-30, each corresponding to the locations of a city
Population size: 3000
Elite size: 50
Selection method: tournament
Tournament size: 4
End criteria: no improvement in fitness for some time
Mutations:

- swap position of two cities (example: A **B** C D **E** F → A **E** C D **B** F)
- rotation of segment of cities (example: A **B C D** E F → A **D B C** E F)
- move position of a city (example: A **B** C D E F → A C D E **B** F)

Input:
Original map30.txt, downloaded from the PTSP Competition homepage.

Output:
Coordinates sequence of the cities in a txt file, which was later used for VECTOR_evolver.

Possible improvements:
Use of some crossovers as described in various books and websites used for classical TSP.
Automated end criteria.
Different representation of the curvature. Could be something with the sequence of paths between cities and angles.

1.2 VECTOR_evolver

An evolver of long number sequence.

The VECTOR_evolver is a classical evolutionary algorithm, with a quite complex fitness function which helped to find better solutions as would some simple vector counting fitness function do. Another thing that contributed a lot to finding a good solution was iterative lenghtening of the path. So at the begging of the evolution of vector sequence we were looking only at two cities and after some generations without improvement of fitness the reading frame was lenghten by one city. And then after next convergence adding another city and so on until we were evolving vector sequence for all 30 cities.

The fitness function:
iSeen+((1-(iUsed/m_iGenomeMaxLenght)))+(2.0/dNearest)+(20/(3+dLastDist));

- *iSeen* is the number of visited cities
- *iUsed* is the number of used vectors
- *m_iGenomeMaxLenght* is the maximum number of vectors for a genome
- *dNearest* is square of the distance of last position before last visited city
- *dLastDist* is square of the distance of last position to the next unvisited city

All factors in fitness function were determined manually with a lot of testing.

Evolutionary algorithm:
Genome: numbers 1-4, representing vector forces
Population size: 10000+
Elite size: 6
Selection method: tournament
Tournament size: 4
End criteria: visited all cities and convergence of fitness
Mutations:

- change vector in genome with probability *p*, where higher *p*'s were less probably, meaning we had more small mutations than large ones. (example: 1 1 2 **3** → 1 1 2 **2**)
- deletition of random vector in sequence (example: 1 **2** 3 4 → 1 3 4)
- deletition of a subsequence of vectors in sequence (example: 1 2 **3 4 1** 2 3 4 → 1 2 2 3 4)
- change of subsequnce to same vectors (example: 1 2 **3 4 1 2** 3 4 → 1 2 **4 4 4 4** 3 4)

Input:
map30.txt from PATH_evolver, with a sequence of city coordinates

Output:
best.txt with vector sequence, that was directly copy/pasted to PTSP submit script.

Possible improvements:
Some crossover between genomes in population.
More detailed fitness function considering some additional distances.
More automation for reruning of program and some other minor modifications to the program.